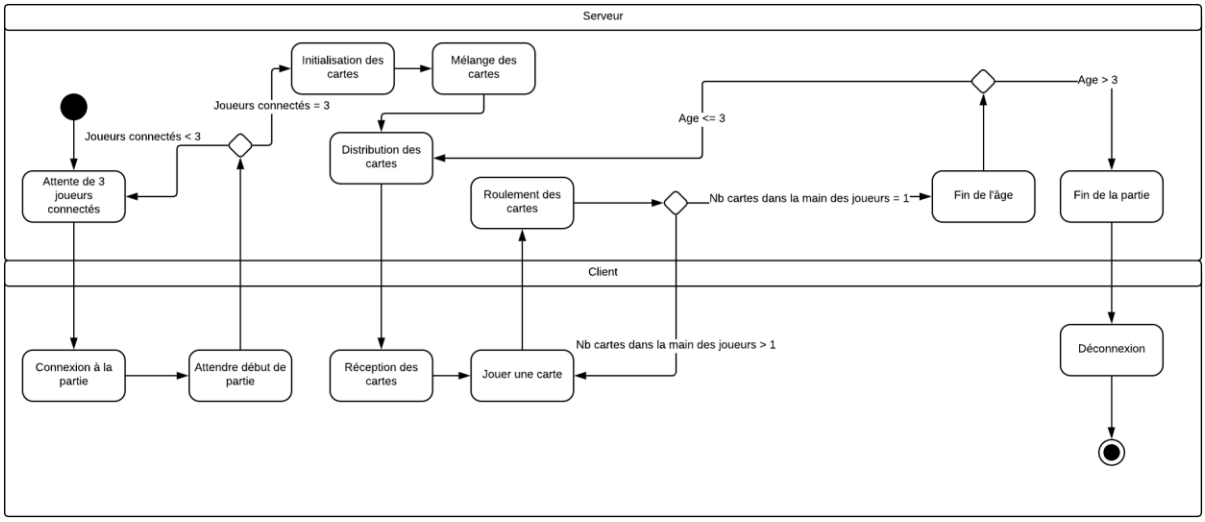
COO
Conception liée au Projet de Développement

Sommaire

Point de vue général de l’architecture

Représentation générale :



Le serveur attend la connexion des joueurs, lorsque 3 joueurs se connectent la partie démarre.
Le client se connecte à une partie et attend que la partie se lance.
Le serveur créer le jeu avec son initialisation et son mélange des cartes.
Puis il distribue les cartes aux joueurs.
Le client réceptionne les cartes et joue une carte.
Les joueurs jouent une carte puis échangent leurs cartes avec le joueur à gauche ou à droite en fonction de l’âge jusqu’au moment ou tous les joueurs ont seulement une seule carte en main.
Si il ne reste qu’une carte à chaque joueur alors c’est la fin de l’âge en cours.
Si l’âge n’est pas plus grand que 3 alors on refait les actions précédentes sinon c’est la fin de la partie et le client se déconnecte.

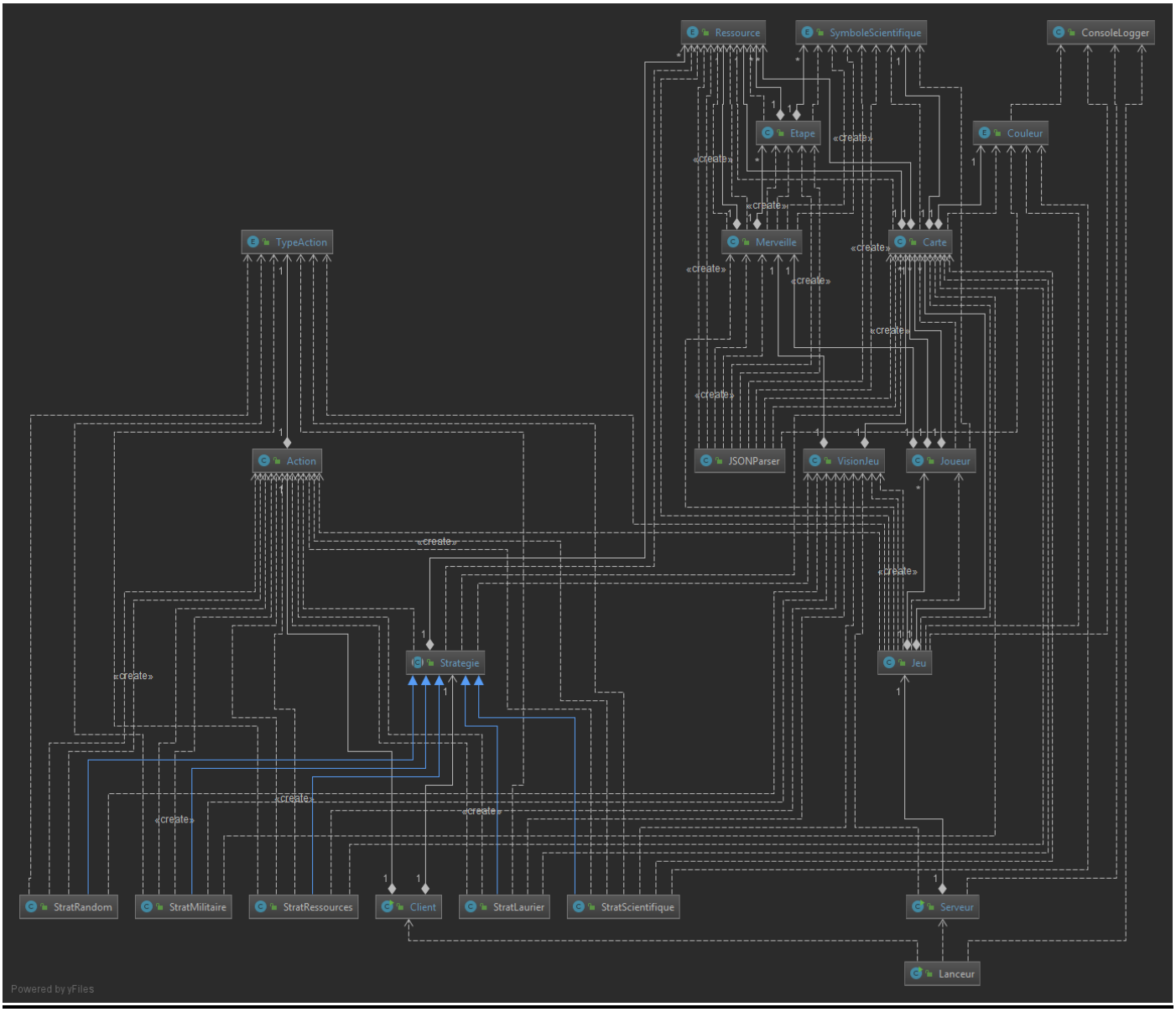
Client

• *Analyse des besoins (exigences) : Cas d'utilisation*
    • Acteurs (le client est un acteur du serveur et vice versa)
    • Diagrammes de Cas d'utilisation
    • Scénarios (sous forme textuelle, un par use case)

• *Conception logicielle*
    • Point de vue statique : Diagrammes de Classes lisible en format A4 (une découpe «logique » avec répétition de certaine classe).

Point de vue Dynamique

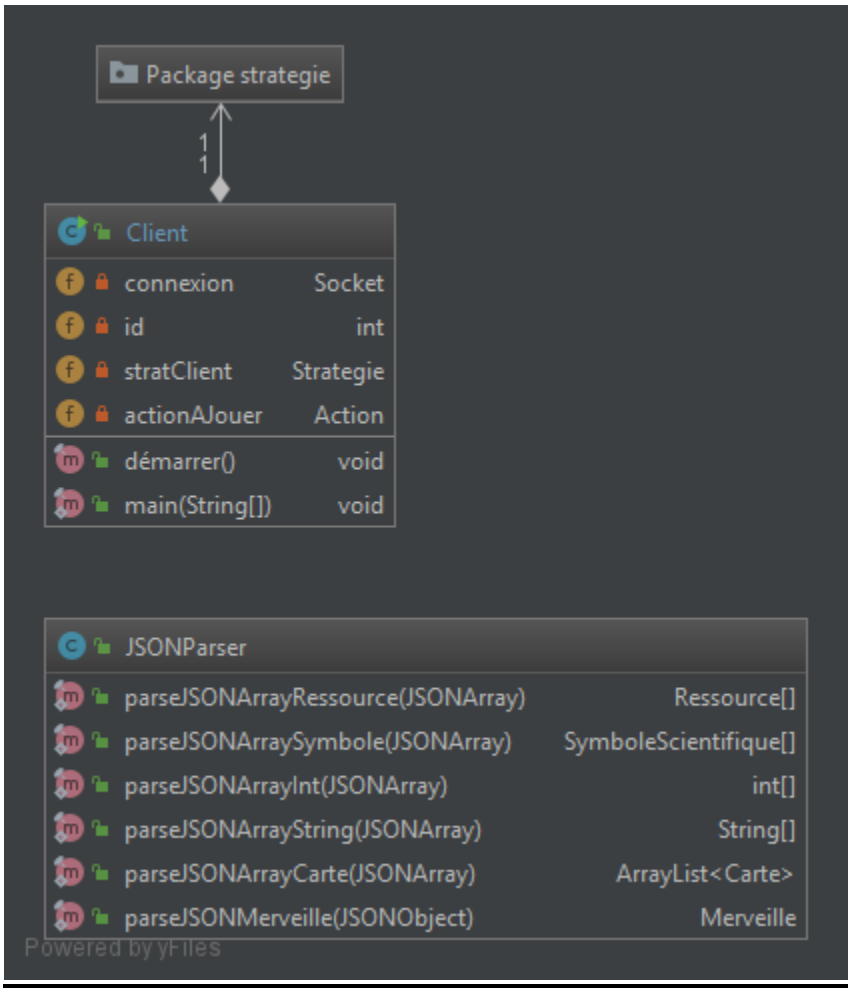
Diagramme de classe général:



Un client ne peut avoir qu'une seule stratégie, et ne peut faire qu'un seul coup (une seule Action) dans un tour. Le serveur n'est lié qu'à un seul jeu, et vice versa. Le jeu contient plusieurs joueurs, et plusieurs decks composés de plusieurs cartes. Celles-ci peuvent

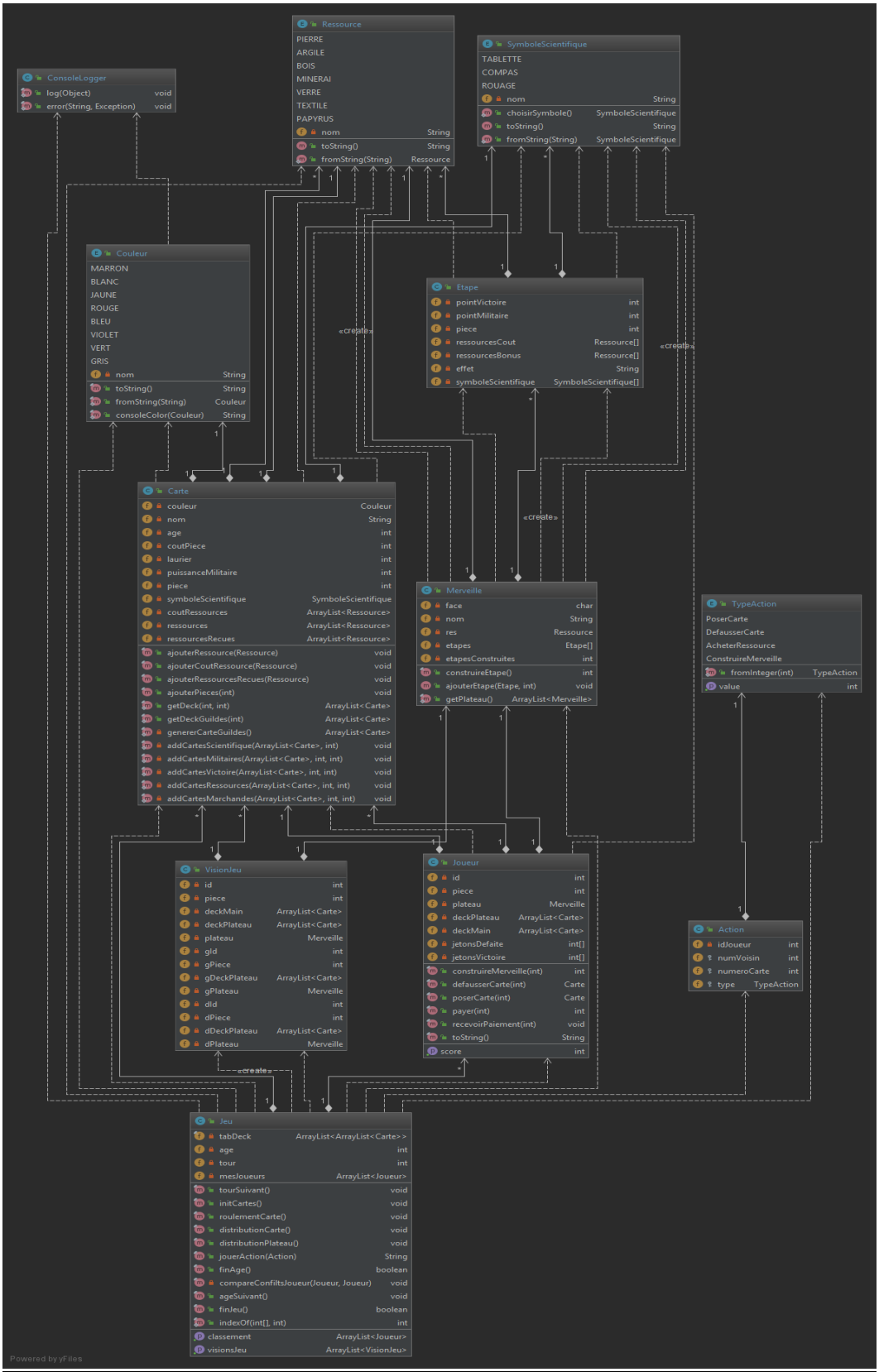
contenir 1 ou plusieurs bonus en ressources, et possèdent un coût en pièce et/ou en ressources. Chaque merveille possède un certain nombre d'étapes (allant de 2 à 4 étapes).

Diagramme du package client:



Un client (ou un bot, dans les faits) doit pouvoir se connecter via une socket. Chaque client possède un id spécifique qui permet leur identification par le serveur. Chacun a une action spécifique à faire à chaque tour, liée à une stratégie particulière.
La distinction en plusieurs stratégies (présentes dans le package indiqué, et sur le diagramme suivant) permet théoriquement une action différente pour chaque client. Un JSONParser est nécessaire pour pouvoir bien traiter l'information transmise avec socket io.

Diagramme du package moteur:



Dans le jeu, il y a différentes ressources: pierre, argile, bois…

Une carte peut posséder différentes caractéristiques: un coût en ressources ou en pièces à l'utilisation, une description, un nom unique et une Couleur. Lorsqu'on la joue, elle peut apporter un bonus variable en pièces, en puissance militaire, en ressources, ou en lauriers.

Selon le déroulement du jeu, une carte peut permettre la construction d'un certain "bâtiment", octroyant certains avantages selon des conditions.

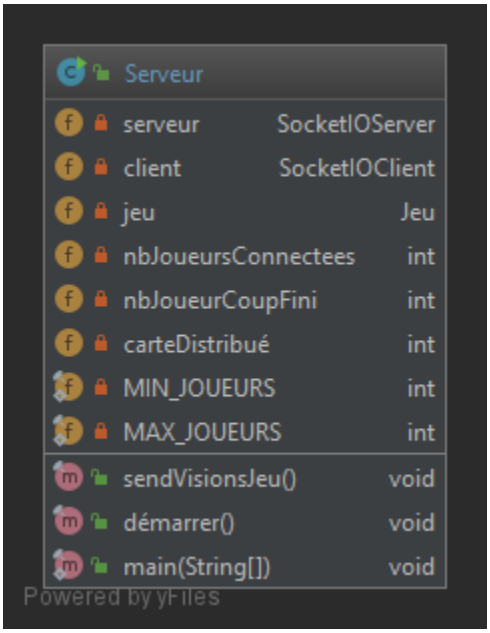
Enfin, plus la partie avance (avec le système d'âges), plus les cartes apportent des effets.

Un Joueur peut posséder une ou plusieurs cartes. Il a un id spécifique, permettant son identification, et possède son propre plateau (Merveille).
Il a également sa propre Vision de jeu, et peut accéder à celles des voisins.
Le joueur possède un certain nombre de pièces, un certain score, et un deck. Après un conflit militaire, il se voit octroyé un ou des jetons de victoire ou de défaite.
Durant un tour, il peut poser (jouer) une carte, ou en défausser une. Si les conditions sont remplies, il peut construire une Etape d'une Merveille.

Un Jeu contient plusieurs joueurs. Il sait combien de cartes il faut distribuer.
D'une manière générale, il fait fonctionner le jeu, c'est à dire: initialiser les cartes, les distribuer, récupérer les cartes afin de faire tourner les decks des joueurs,
faire jouer un joueur, et comparer les scores afin d'établir le classement général. La méthode "indexOf", quant à elle, nous permet de retrouver une certaine carte dans le deck d'un joueur.
Le Jeu doit aussi avoir accès aux decks des joueurs, et aux joueurs eux-mêmes. En fonction du Type de l'Action, le Jeu effectuera une certaine Action. A la fin d'un âge, le Jeu se charge des conflits militaires entre les Joueurs.

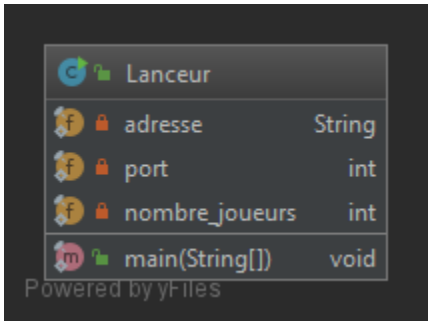
Enfin, nous utilisons JSON afin de faire circuler correctement les informations dans la liaison client-serveur.

Diagramme du package serveur:



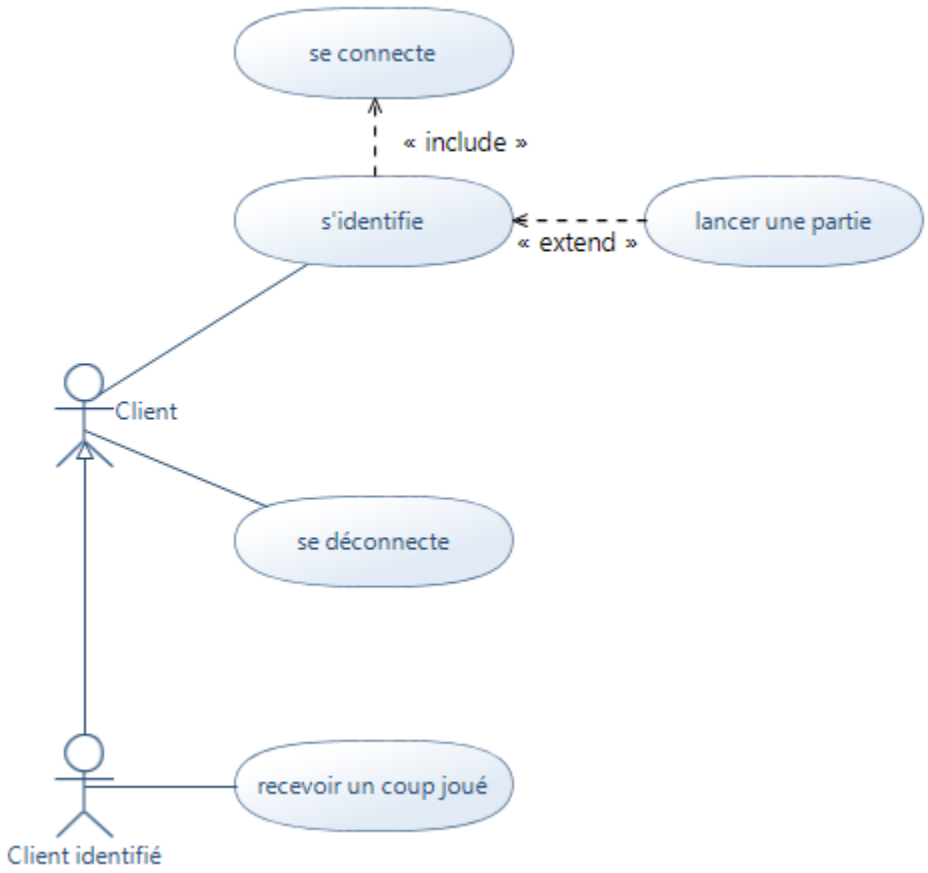
Notre serveur doit interagir avec les clients: contrôler le nombre de joueurs connectés afin de pouvoir commencer une partie, envoyer les decks aux joueurs, signaler le début et la fin des tours, des âges et du jeu avec toutes les opérations qui s'en suivent.

Diagramme du package lanceur:



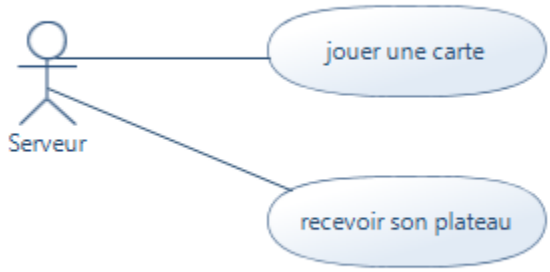
Le Lanceur doit lancer le serveur afin que les joueurs puissent s'y connecter. Il crée également nos joueurs (les bots), selon un nombre de joueurs qui lui est indiqué.
Pour faire tout cela, il doit posséder une certaine adresse pour lancer le serveur à cette adresse, et un port spécifique.

Use case Client :



Le client peut se connecter et se déconnecté du serveur, la connexion lui permet de rejoindre de la partie (une fois le nombre de joueur requis atteint). Suite à cela, le client peut observer les bots jouer une partie ainsi que le déroulement de la partie (les bots reçoivent des cartes, jouent des cartes, …).

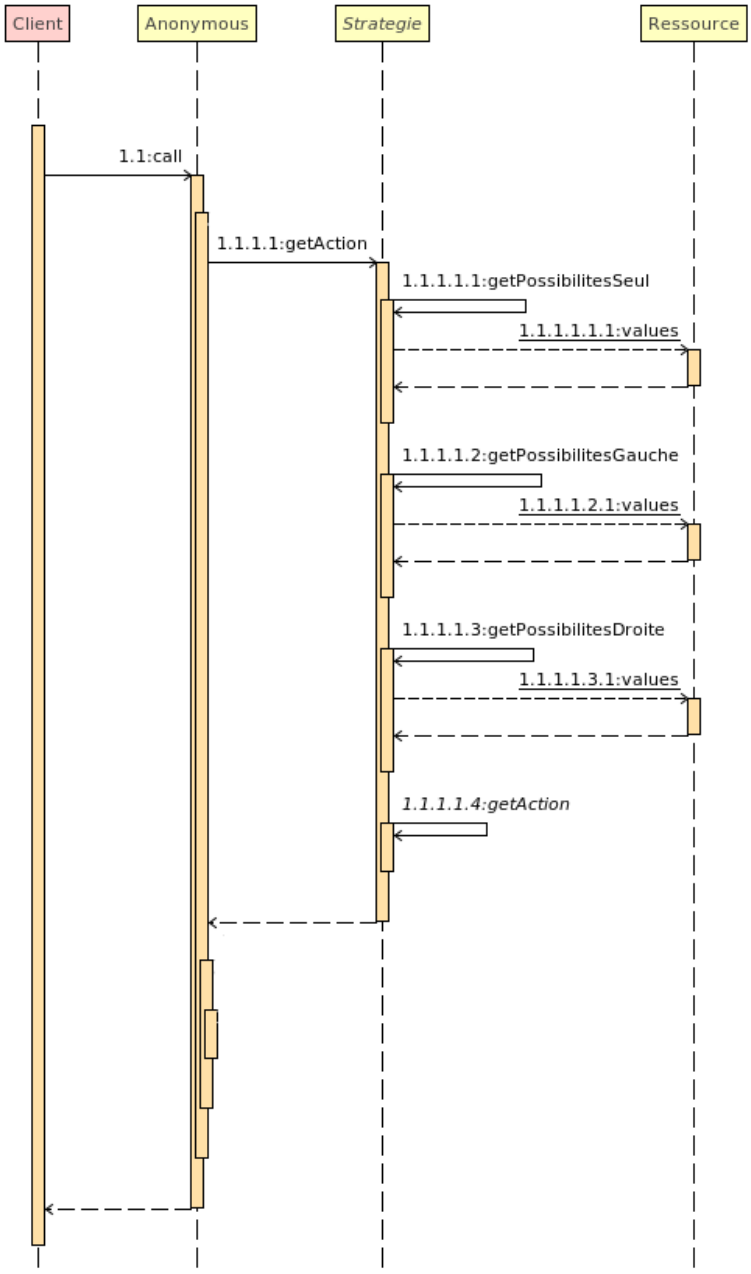
Use case Serveur :



Le serveur permet (au client abonné) de recevoir un plateau et de pouvoir intéragir avec le jeu, autrement dit “jouer une carte”.

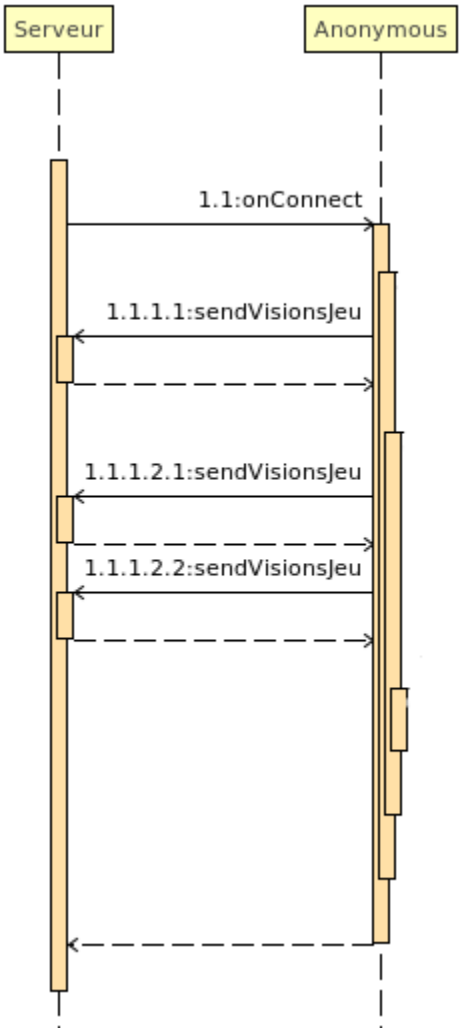
Diagramme de séquence:

Client:



Le client demande quelle action réalisé à la classe Stratégie. Stratégie choisis une stratégie à adopter en fonction des Ressources de ces voisins de gauche et de droite.

Serveur:



Le serveur communique avec le le code exécuté à la réception des requêtes client en lui communiquant la vision du jeu.

Conclusion

Points forts :

- Structuration du projet
- Client/Serveur sur un seul thread

Points faibles :

- Quelques problèmes (mineurs) de conception

Évolution prévue :

- Finalisation du projet
- Amélioration des stratégies